2002 autumn

百萬石

chofu-fest edition

部長挨拶

MMA 部長 佐野 浩司 (sano@mma.club.uec.ac.jp)

2002年11月

本日は、MMAにお越しいただきありがとうございます。

MMAは"Microcomputer Making Association"の略で、1970年代にその歴史を遡ります。以前はMMAの名前の通りにコンピューターを製作していたようですが、最近はパソコンの高性能か、低価格化に伴い、活動の中心はハードウエア製作から、UNIXを中心としたネットワーク関連の研究、ソフトウエア開発にシフトしてきました。

そのような流れの中、去年に引き続いてプログラミング能力の向上を追求すべくACMの主催する国際大学プログラミングコンテスト(ICPC)に参加し、国内予選を突破、金沢大学でのアジア地区予選に進出することができました。

今年の調布祭において、MMAは1、2年生による展示、恒例のジャンク屋等を行いますので、どうぞお楽しみ下さい。

ICPC体験記番外編、Judgeシステムの裏を暴け！

飯村 卓司 ⟨ takkun@mma.club.uec.ac.jp ⟩

2002年11月

1 abstract

MMAで鳴らした俺達ICPC選抜チームは、Judgeシステム、通称 PC^2 に慣れる時間を与えられた。しかし、普通に正解の答えを出せるのを確認するだけで満足するような俺達じゃぁない。知識欲さえ満たせばなんでもやってのける恥知らず、非常識を実行し、楽しかったと言い放つ、俺達、最低野郎MMA-NFX!

俺は、リーダーtakkun。通称uirou。最初のインスピレーションだけで全てをごり押しで解く。
俺のような万年留年生でなければ、こいつらみたいな未来のない者共のリーダーは勤まらん。

俺は入学してからまだまだ三年目。通称xmoe。自慢のSTLテクニックで、問題なんてイチコロさ。
template使って、サービス問題から幾何問題まで、何でも解いてみせるぜ。

ようお待ちどう。俺様こそniku。通称クレイジーniku。ルーレットでの運は天下一品!
奇人？変人？だから何。

現役院生。通称ikidou。チームのコーチだ。全ての悪事を黙認してやらぁ。でも、犯罪だけはかんべんな。

俺達は、道理の通らぬ世の中にあえて挑戦する。頼りになる神出鬼没の、最低野郎MMA-NFX!
助けを借りたい時は、いつでも言ってくれ。

2 目的と PC^2 の概要

PC^2[1]は、Programming Contest Controlシステムの意味で、ぴーしーすくうぇあ、またはぴーしーつーと呼ばれる、プログラミングコンテストにおいて問題に対する答えのソースを提出させ、それを判定するためのシステムである。我々コンテスト参加者は、このシステムのクライアント部分である、答えのソースを提出し、その結果、正解であったかどうかを得るためのソフトのみを使用することになる。そこで問題になるのが、機械に判定される、という事である。これが人間が相手に存在するのであれば、多少の誤差のような事も、「常識的に」判定されるものと考えられるが、相手が機械では、どういう事がちゃんと処理出来て、どういう事が処理できないか、というのを把握している必要がある。そこで、まずは PC^2 で扱う事の出来る事を確認してみる。

まず、こちらから指定できるのは、

- ソースファイルの場所
- 他に必要なソースファイルの場所のリスト
- 問題番号
- 使用するコンパイラ

である。いたって普通である。次に、問題と解答の形式を確認しておく。問題は、入力が一つのファイル、出力は標準出力を利用する事になっている。入力されるファイルの名前は、問題文中に書いてある。また、入力されるファイルのフォーマットは、テキストファイルで、だいたいは一行目にデータ数、その後の行にデータが記述されている形になっている。そして、出力しなければならないフォーマットは、英文の文章で指示されており、だいたいにおいては、「～の数を出力せよ」だの、「～の面積を出力せよ」といった形式で指示されている。これからわかるように、入力の形式に比べると、どうにも出力の形式の指定が甘い。ここはいろいろと実験すべき項目である事が容易に想像できる。

次に、PC^2 が返してくるメッセージにはどういったものがあるのかも確認しておく。PC^2 は、以下のようなメッセージを返すということになっている。

- YES 問題に正解した場合
- NO 答えが正しくない場合
- NO 時間切れ (180秒以上は時間切れとの説明がある)
- NO 答えに余分なものがついていた場合
- NO 内部エラーで落ちた場合
- NO 予期しないエラーなので、スタッフを呼べというエラー

[1] http://hera.ecs.csus.edu/pc2/

等である。基本的に printf デバッグなどは出来ないというわけである。
あと、いくつかの疑問点も出てきた。

- コンパイラにオプションは渡せるのか
- 明らかに間違えた解答を出した場合、どのくらいの時間で間違いと判定されるのか
- SIGSEGV などを起こさせた場合、致命的な問題は起こらないのか
- etc...

である。我々にはこれらをきちんと把握しておく必要がある。そう。その必要があるのだ。

3 実験方法

実験はコンテストの前日の、トライアルの時間を用いて行われた。このトライアルの時間はたったの2時間半、これだけの時間で我々の要求する全ての実験を完了しておかねばならない。そのためにはきちんと戦略を考えてから、実験する必要があった。
まず、正解の解答を作り、それに変更を加える事で、正しい解答となるか、ならないかで実験を行う。これをすることにより、正しくその行為が容認されているか、いないか、がわかることと思う。そう。思っていたのだ。

4 結果

さて、我々は上記の考察の下に、以下に述べる実験を行った。これから書かれることはフィクション無しの真実であり、この行為が PC^2 の実力を確かめ、予期しない動作を出来うるかぎり実験し、その潜在的な問題点を浮き彫りにする事を目的にした行為である事をここに明記しておく。

4.1 普通の実験

まずは、正しい答えを出すためのプログラミングである。トライアルで配られた問題は3問。そのうちの一つは一行づつ数字が記述されるので、その数字の合計を表示するというものである。もちろん、こんなものは一瞬でプログラミング可能でああるから、我々はさっさとこれを解き、次なる実験へと駒を進めたわけである。ただ、このYESが出てくるまでの時間は、3分程度の時間がかかった。案外 PC^2 も遅いものである。もしかしたら、YESの場合は、人間に処理を渡すような事が行われているのかもしれない。

TEST: “正しい答えを出した場合”
Result: YES

4.2 間違えてみる

次に、普通に間違えた答えを提出した場合、の動作を実験する。間違えた答えの出し方にはいくつかある。簡単にまとめるとこうだ。

- 数字の答えなのに文字列である等、明らかに間違えている
- output format では指定されていない、解答の前後に改行や、空白文字列を入れてある
- 何も出力しない
- 180秒以上何もしないで時間切れを誘う
- 大量に何かを出力する

それぞれについて実験してみた時の解答例は以下の通り。

- `printf("We are MMA!");`
- `printf("  %d  ", answer);` や、`printf("%d\r\n", answer);`、`printf("%d\r", answer);` 等
- `main(){return 0;}`
- `sleep(181); printf("%d\n", answer);`
- 実験せず

それぞれ、"\r\n"を改行としたもの以外、全て期待通りの不正解にされた。すなわち、ほぼ全てが間違えた答えであると判定され、時間切れは正しく時間切れと判定された。ある意味期待通りであり、"\r\n"も改行として許されるという少し賢いシステムであることも判定できた。また、間違いがどのくらいの時間で判定されるかという事も測定できた。だいたい間違いであると判定されるのには30秒近くかかった。これは、本当に相手が機械であるのかどうか激しく疑ってしまう位遅いものであるが、堪え切れないほど遅いわけでもないというのがわかっただけでも意味があるだろう。

TEST: “間違えた答えを出した場合”
Result: NO 少し時間がかかる

4.3 怪しく間違えたりしてみる

次に少し普通でない状態で間違えた場合の事について実験してみる。簡単に言うとそれだけであるが、これは実際にそのような間違いが起こった時に、どういう反応を PC^2 が示すのかを把握するためには重要な事である。

この実験では以下の項目がテストされた。

- SIGSEGV するプログラムを書いてみる
- 0 以外の終了コードでプログラムを終わらせてみる
- libm が必要な、sin(3) を用いるようなプログラムを書いてみる

この場合、SIGSEGV するプログラムはただ単に NO、間違いであると判定され、0 以外の終了コードで終了した場合には、YES、正しく正解と判断された。要するに終了コードは問題にはならないという事である。また、libm が必要なプログラムは問題なくコンパイル、実行されたらしく、YES、正しく正解と判定されていた。このことから、コンパイルオプションは指定できないが、実用上問題ないコンパイルオプションが指定されている事がわかった。

TEST: "怪しく間違えた場合"
Result: NO 等 予想通り

4.4 PC^2 に挑戦してみる

さて、ここからが本題である。我々はコンピュータでのプログラムを扱うにあたって、常にセキュリティに心を配るべきである。もちろん、PC^2 も例外ではない。我々はこの PC^2 のセキュリティを正しくテストするべきである。そういうわけで、我々は以下の実験を行った。

- fork(2) が使えるかどうか試してみる
- ファイルを新しく作成し、それをアクセスできるか試してみる
- 一度ファイルを作製するプログラムを送り込んでから、それと同じ名前のファイルを開く別のプログラムを送り込んでみる
- exec(2) が出来るかどうか試してみる
- ベルコードのような、表示されないのだけれどデータとしては効果のあるものを提出してみる

これらについてはそれぞれについて解説を入れよう。

1 fork(2) は使えるか

これについては簡単な無限 fork(2)… ではなく、正しく fork(2) して子プロセスが答えを提出するプログラムを作成した。これは問題なく動作し、我々は fork(2) も許されている事を確認したのである。

TEST: "fork(2) は使えるか"
Result: YES 使えてしまった

2 ファイルは作れるか

ファイルを作れるとアルゴリズムが変わる。これはとても重要な概念である。そこで我々はファイルを用いた演算も出来るのかをチェックした。やり方は簡単で、一度答えをファイルに書きだし、それを読み込んでから、またそれを答えとして出力するプログラムを書いた。

これも問題なく動作し、我々はファイルを使ったアルゴリズムも利用できる事を確認したのである。

TEST: "ファイルは作れるか"
Result: YES 作れる

3 前に作られたファイルにアクセス出来るか

そこで我々に一つの疑問が生じた。それは、「前にファイルを書き出しておいて、後で使えるととても便利なのでは」ということである。これはさっそく実験された。すなわち、先程のプログラムが作ったファイルを、次に送り込んだプログラムで開いてみて、それが開けたら正しいプログラムを動かし、開けない場合にはそのまま exit(3) するというものである。

これは NO、間違いという答えが帰ってきた。非常に残念であるが、我々はファイルを残すことは出来なかった。

TEST: "ファイルは再利用できるか"
Result: NO さすがに再利用は出来ない

4 exec(2) は使えるか

さて、世の中には星の数ほどのプログラミング言語が存在する。ICPC/ACM プログラミングコンテストにおいては、C,C++,Java,Pascal という言語を用いてプログラミングの腕を競う事となっている。しかし、である。C と C++の間に STL 等というライブラリレベルでの格差があるにもかかわらず、それに対する救済措置は存在しない。さらに、我々は Perl の扱いに長けており、C なんぞで書くよりも Perl を用いて書いた方が何倍も生産性が上がる問題というものも存在するのである。しからば、我々は Perl が使えるかどうかを試してみるべきであろう。というわけで、我々は以下のコードを正解のプログラムの最後に潜ませ、Perl が動作するかどうかを確かめたのである。

```
{
  char str[1024];
  sprintf(str, "print \"%d\"", answer);
  execlp("perl", "perl", "-e", str, NULL);
}
printf("wring answer\n");
```

このコードで YES、すなわち正しい答えであるとの判定が下されたなら、我々は Perl を手に入れたことになる。

しかして、結果は YES であった。ここに、我々は Perl を手に入れたのである[2]。なお、このテストを行った後、この exec(2) システムコールを使った部分を、以下のように書き換え、テスト用に送りつけるコードの中に、成功例として残すこととした。

```
/***** WARNING!!! SEE THIS!!! ******
{
  char str[1024];
  sprintf(str, "print \"%d\"", answer);
  execlp("perl", "perl", "-e", str, NULL);
}
*/
```

これが、後で楽しい事を引き起こすことになる。

TEST: “exec(2) は使えるか”
Result: YES 使えてしまった

5 ベルコードを送るとどうなるか

次に我々がテストしたのは、ベルコードを出力して Judge システムの端末でベルを鳴らすという事であった。考えてもみたまえ。ネットワークの先にある、Judge マシンから不思議と beep 音がするのだ。こんなにシュールな光景があろうか。いやない。これは是非にでも実行してみないわけにはいかない。

ということで我々は以下のようなコードを PC^2 に送り込んだ。

```
{
  int i;
  for(i = 0; i < 10; i++){
    printf("\b"); fflush(stdout);
    sleep(1);
  }
}
printf("%d\n", answer);
```

これで、Judge システムの端末から、10 回、意味不明な beep 音が発せられるはずである。

我々はわくわくしながらその時を待った。

一分、

二分、

ちっとも音は鳴ったように見えない。あぁ、やっぱり Judge システムはこのホールでない何処か別の場所にあったのか。と、思ったその時。判定が帰ってきた。

NO 予期しないエラーなので、スタッフを呼べ

……何か問題でもあったのだろうか。このエラーは我々を困惑させた。その時、大会運営側の人間、すなわち本物の Judge から一枚の紙がやってきた。そこには、あせったような殴り書きで、こう記されていた。

Team 25
This is a C/Java/Pascal
programming contest!
Do not use exec system call.
This can incur disqualification.

[2]本戦では使いませんでしたよ。そりゃもちろん。

もちろん、Team 25 とは我々のチーム No の事である。

素晴らしい

なんと素晴らしい。PC^2 の向こうには、人間とも見紛うばかりの賢い知性が存在したのである。いや、これは正しく人間が存在する証明であろう。

我々はこう推理した。

1. PC^2 は、問題に対する答えを識別する能力がある
2. しかし、人間もその答えをみる
3. 今回の場合、人間が答えをみると、普通にはベルコードを見ることができないので、正しい答えが出ているように見える
4. しかし、PC^2 はベルコードを識別し、この答えは間違いであると判定する
5. 人間の Judge は、なぜ正しい答えが間違いと認識されているのかわからないので、該当の source を眺める
6. WARNING!!! SEE THIS!!! と書かれた、exec perl の行を見る

という、経緯ではないだろうか。とすれば、ベルコードの実験を行った後、すぐに、この紙を渡されたことにも頷く事が出来る。もちろん、真実は闇の中であるが…

TEST: "ベルコードを送るとどうなるか"
Result: NO 人間の Judge に目をつけられる

5 心残り

また、出来れば以下の項目についても実験したかったのだが、時間切れで実験出来なかった。

- fork(2) と socket(2) を用いて、daemon を作成し、我々の PC との connection を確立し、shell を奪う
- どのくらいのファイルサイズまで作れるのか。巨大ファイルサイズのファイルを作ったら、他のプログラムを送り込んだ参加者を締め出せるのか
- 無限 fork(2) を用いて、Judge システムの CPU Power をどこまで食いつぶせるか

これらの事を行えば、我々MMA-NFX が勝利を収めるための強力な weapon になるのは間違い無い。これは今後の課題にしたいと思う。[3]

6 本当の真実

以上の実験を行い、我々は PC^2 の穴、そして使い方を理解した。そして、本戦においては一度も NO を出すことなく、正しく YES のみを叩き出し、全てストレートで正解した。もちろん、本戦の直前のアナウンスにおいて、

> 昨日のテストの時に、まずいことをやった奴が居る。本戦でやったら即失格ものの事であった

と、アナウンスされた事は我々は誇りに思う。[4]

そして大会後の表彰式後のパーティにて、Judge をしておられた G アン先生にお話を聞く機会を受けた。そこでわかった事が以下である。

- PC^2 の先には人間が居る
- PC^2 によって YES の判定がなされた後には、人間がそのソースを眺めてから、YES の判定を出す
- もちろん、NO の判定がなされたものについても、人間がそれを見て、救いだすことがある
- Judge のメンバは、どの大学のどのチームが、Team 25 であるかを認識していない
- そのため、間違いの答えを書く時に、決して `printf("We are MMA!\");` などと書かないこと。Judge は手心を加えてはいけないのである
- もちろん、exec(2) システムコールを使うなんてのはもってのほかである。オマエは真面目にやる気があるのか
- Judge の心象が悪くなると、いろいろと不利である

要約すると、

> PC^2 は機械だけでなく、人間も組み込まれている。からかうような事をすると Judge の心象が悪くなる。目をつけられるとやはりマズいので、そんなことするな。

もちろんである[5]。

---

[3]……思ってませんヨ。
[4]……思ってませんってば
[5]もし本戦で不正を働きたいのなら、トライアルの時間にわざわざ目を引くように悪い事なぞしない。ちゃんと隠れて行うに決まっている。

7 最後に

今回のICPC/ACMプログラミングコンテストのトライアルにおいて、我々は有意義な実験を行えた。これでわかったのは、常に相手の事を思いやりつつ行動する事が、先の自分のためである、という事である。

また、我々のアホな実験を黙認してくださったJudgeの方々には頭が上がりません。ここに我々は深い謝罪の意を表すとともに、二度と同じ実験はしない事を明記するものである。

J2ME for Palm 私的メモ

佐野 浩司 ⟨sano@mma.club.uec.ac.jp⟩

2002 年 11 月

概 要

1 はじめに

J2ME とは Java2 Platform, Micro Edition の略で、簡単にいえば PDA 機器や携帯電話で Java を動かしてみようというものです。J2ME には CDC(Connected Device Configuration ネットワーク情報機器用)、CLDC(Connected, Limited Device Configuration CPU 能力やメモリ容量が制限された携帯型ネットワーク情報機器用) の 2 つのライブラリ構成があり、CLDC 仕様は様々な製品に共通する基本ライブラリ仕様で、ユーザーインターフェースなどの詳細な仕様は MIDP(Mobile Information Device Profile) で規定されています。他の Java2 プラットフォームの仕様には、J2EE(Java2 Enterprise Edition)、J2SE(Java2 Standard Edition) がありますね。

Palm 機上の開発環境なら CodeWarrior や、PRC-Tools がありますが、「お金を掛けたくない」、「手軽になんかやりたい」、「Java はやったことがある」みたいな人は J2ME を試してみるとよいでしょう。(ちなみに筆者は上記の人間ですが Java はさらっとしか触ったことはありません)

2 必要なもの

とりあえず筆者が使ってる Palm の情報の同期をしているマシン（母艦と呼ぶのがいまどきのトレンドらしいぞ！）が Windows 環境なので、それを前提にして話をすすめます。

POSE(Palm OS Emulator)

いちいち HotSync でつくったプログラムを Palm 機に移しかえるのは面倒なので、開発マシン上でエミュレートするのに使用します。ちなみにインストールしただけでは動かず、Palm OS の ROM イメージが必要です。ROM イメージを入手するには Palm 社と契約をして書類を書いて云々する方法もありますが、面倒なので実機から取り込みます。POSE に付属する ROM Transfer.prc を使用するとクレードルから Palm OS を転送することが出来ますが、シリアルポートからしか転送できませんので、USB を使用する最近の Palm 機を持っている人は別にシリアルケーブルを買う必要があります。Palm Desktop のフォルダにある USB 関連の DLL を POSE のフォルダにコピーしてみるとあら不思議、通信手段の選択肢に USB が、という手もあるようですが、筆者が試したところ実現しませんでした。ということで今回筆者は Palm m100 の ROM イメージを使用しました。

`http://www.palmos.com/dev/tools/emulator/`

MIDP for Palm OS(Mobile Information Device Profile for Palm OS 1.0)

Palm OS のための Java ランタイム環境。といっても見た目は単なる Palmware なのでインストールも削除も簡単。下記のサイトから落としましょう。ちなみにデモ用にいろいろソフトが入っているみたいなので暇なときはいじってみましょう。

`http://java.sun.com/products/midp4palm`

J2SE(Java2 SDK, Standard Edition)

Java アプリケーションの作成、構築環境ですね。下記から落とします。

`http://java.sun.com/j2se/1.4/ja/download.html`

J2ME Wireless Toolkit

「CLDC および MIDP に準拠する携帯電話やエントリレベルの携帯情報端末のための Java テクノロジアプリケーションを開発するのに必要な、エミュレーション環境、マニュアル、およびサンプルアプリケーションを備えたアプリケーション開発者向けのツールセット」だそうです。下記のサイトでは日本語版が落とせます。

`http://java.sun.com/products/j2mewtoolkit/ja_download.html`

J2ME Wireless Toolkit Japanese Documentation 1.0.3

J2ME Wireless Toolkit の日本語版ドキュメントですね。下記から落としましょう。

`http://java.sun.com/products/j2mewtoolkit/ja_download.html`

それぞれのファイルをインストールすると、J2ME の開発環境がそろいます。

3 なんか作ってみる

J2ME Wireless Toolkit 1.0.3 の、KToolbar を起動させます。新規作成 (N) を選び、プロジェクト名とクラス名にそれぞれ TestPalmApp、HelloWorld と入力します。プロジェクトの作成を選ぶと、属性を設定するウインドウが開きますが、とりあえず了

解を選んでおきます。そうすると新しいプロジェクトが作成され、apps フォルダの中にプロジェクト名と同じ名前のフォルダが作成されます。ディレクトリの中は、下記のようになっています。

```
├ TestPalmApp
│   ├ bin
│   │   ├ TestPalmApp.jad
│   │   └ MANIFEST.MF
│   ├ lib
│   ├ res
│   └ src
```

Java のソースファイルは src フォルダの中に作成します。今回は以下のようなファイルを作成します。

リスト 1: HelloWorld.java

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet {
  public void startApp() {
    Form form = new Form("Hello World!");
    form.append(new StringItem("Message", "Hello World!"));
    Display.getDisplay(this).setCurrent(form);
  }
  public void pauseApp() {}
  public void destroyApp(boolean unconditional) {}
}
```

import で MIDlet クラス、javax.microedition.lcdui パッケージの使用を宣言しています。extends MIDlet により HelloWorld クラスの定義で MIDlet クラスを継承しています。Palm で動作する Java プログラムは、必ずこの MIDlet を継承します。また、startApp() メソッドはプログラムが開始する時、 pauseApp() メソッドはプログラムが一時停止される時（たとえば携帯電話上でプログラムを動かしているときは電話がかかってきたとき）、destroyApp() メソッドはプログラムが終了する時に呼び出されます。startApp() の内部では、まず new Form("Hello World!") において new 命令で部品 Form のインスタンス (オブジェクト) を生成します。引数として与えた文字列が Form の名前として表示されます。append メソッドで文字列を含むことができる StringItem を追加しています。Display.getDisplay(this) で、this はこのアプリケーション自身を意味しますから現在の画面を示す Display インスタンスを得ることができます。得られたインスタンスに対し setCurrent(form) と form インスタンスを指定すれば、form が画面に表示されます。

それでは KToolbar のビルド (B) を選びましょう。コンパイルされます。コンソールに「ビルドは完了しました」と表示されれば問題ないでしょう。ディレクトリの中は、下記のようになっているでしょう。

```
├ TestPalmApp
│   ├ bin
│   │   ├ TestPalmApp.jad
│   │   └ MANIFEST.MF
│   ├ classes
│   │   └ HelloWorld.class
│   ├ lib
│   ├ res
│   ├ src
│   │   └ HelloWorld.java
│   ├ tmpclasses
│   │   └ HelloWorld.class
│   └ tmplib
```

実行させるには、デバイスを PalmOS_Device に変更し、実行 (R) を選択します。すると、TestPalmApp.prc が作成され、POSE が起動し作成したプログラムが実行されます。実行すると、画面にきちんと Hello World!と表示されますね。

表示したら表示しっぱなしというのもなんなので、終了ボタンをつけます。

リスト 2: HelloWorld.java

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet implements CommandListener {
  private Command exitCommand = new Command("Exit", Command.ITEM, 1);
  public void startApp() {
    Form form = new Form("Hello World!");
    form.append(new StringItem("Message", "Hello World!"));
    form.addCommand(exitCommand);
    form.setCommandListener(this);
    Display.getDisplay(this).setCurrent(form);
  }
  public void pauseApp() {}
  public void destroyApp(boolean unconditional) {}

// implements CommandListener

  public void commandAction(Command c, Displayable s) {
    if (c == exitCommand)
      notifyDestroyed();
  }
}
```

implements CommandListener で終了というコマンドが起動されたことを知るためのリスナー・インターフェースを実装します。また、form オブジェクトに対し、setCommandListener() でコマンドイベントを受け取ることを指示します。commandAction() メソッドで受け取ったコマンドが exitCommand なら、notifyDestroyed() メソッドでプログラムを終了させます。

4 まとめ

かなり早足で紹介しましたが、なんとなく J2ME がどんなものかがわかったと思います。作成されたプログラムも軽快ではないし、なにより JVM がでかく、なんだか中途半端なイメージですが、わりと手軽に開発が出来ることがうれしいことだと思います。これでごりごりプログラムを書いて、お友達同士赤外線でプログラムを飛ばしあいましょう。

参考サイト：Palm Hackers Salon

http://salon.simple-palm.com/

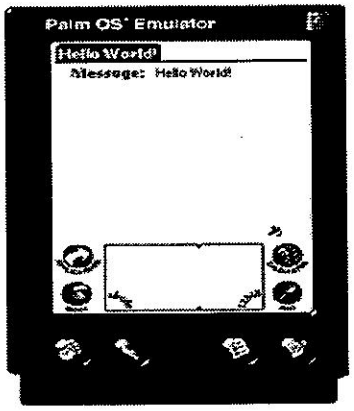

図 1: Hello World

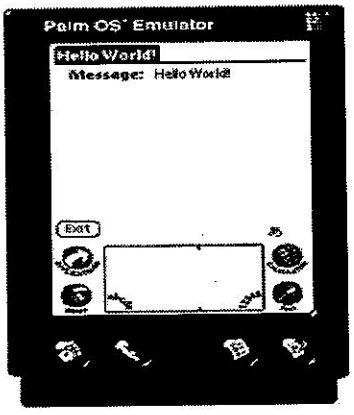

図 2: Hello World (終了ボタンつき)

IP Filter によるパケットフィルタリング

谷島 ひでのり 〈 yajima@mma.club.uec.ac.jp 〉

2002 年 11 月

1 導入編

FreeBSD を前提に書いていますが、 Solaris, IRIX,*BSD 等でも利用できます。
ただし OpenBSD ではライセンスの関係で標準では組み込まれていません。代わりに PF があるようです。

IP Filter を有効にするにはカーネルコンフィギュレーションファイルに以下の行を加えて、カーネルを再構築します。

```
options         IPFILTER
options         IPFILTER_LOG
```

これで IP Filter が有効になります。2 番目はログを取るために必要です。

```
options         IPFILTER_DEFAULT_BLOCK
```

これも加えておいた方が良いと思います。これだとデフォルトですべてのパケットがブロックされるようになります。
不必要なポートを塞ぐより、必要なポートを開けていく方が安全です。
あとは /etc/rc.conf に以下の行を加えます。ここではルールファイルを /etc/ipf.conf に置くことにします。

```
ipfilter_enable="YES"                   <-- IP Filter を有効にする
ipfilter_rules="/etc/ipf.conf"          <-- ルールファイルを指定
ipmon_enable="YES"                      <-- ログを有効にする
ipmon_flags="-D /var/log/ipf.log"       <-- ログを記録するファイルを指定
```

これで再起動すると IP Filter が有効になります。
再起動させたくない (できない) 場合は、カーネルモジュールを組み込みます。

```
#kldload ipl
```

これでまず侵入されることはないでしょう。どこへも、どこからも全くアクセスできません。ただこれだとネットワークにつながってないも同然なので、少しづつ穴をあけていきます。

2 ルールの作成

これはあらゆるパケットが自由に出入りできるルールです。

```
pass in from any to any
pass out from any to any
```

まずはじめにアクションを書きます。

- pass パケットを通過させるようにする。
- block パケットを叩き落すようにする。

また in か out で入って来るパケットか出ていくパケットかを指定する必要があります。
次に書いてあるのはマッチングパラメータです。

- from address/mask 送信元を指定します。
- to address/mask 送信先を指定します。

any は全 IP アドレスにマッチします。
また from any to any は all で代用できます。
ルールファイルを書いたら、

```
#ipf -Fa -Z -f /etc/ipf.conf
```

これでルールが有効になります。ちなみにルールファイルは最後に書いたものが有効になります。

この状態で正常に通信できたなら、まともなルールを書いていきます。ここでは外部につながるネットワークインターフェースを xl0 とします。

とりあえず、自分自身・外部に出ていくパケットは素通しにしてみます。

```
pass in on lo0 all
psss out on lo0 all
pass out on xl0 all
```

TCP は双方向の通信をするので、戻りのパケットを受け入れる必要があります。

```
pass in on xl0 proto tcp all flags A/A
```

これで TCP では外部と通信できるようになります。

- proto PROTOCOL プロトコルを指定します。プロトコル番号を 10 進数で記述するか、/etc/protocols のプロトコル名が使えます。
- flags FLAG1/FLAG2 TCP でのみ有効です。TCP ヘッダで指定可能なフラグを記述します。

```
F - FIN
S - SYN
R - RST
P - PUSH
A - ACK
U - URG
```

パケットのフラグを FLAG2 でマスクして FLAG1 と比較します。

```
flags A
```

とした場合は、フラグに ACK が含まれていればマッチします。 flags A/AUPRFS と同じです。

```
flags A/A
```

の場合は、フラグが ACK のみの場合にのみマッチします。
名前で引けないと嫌なので、DNS の応答を受け入れます。

```
pass in on xl0 proto udp from any port = 53 to any
```

- port = PORT ポート番号を指定します。from の次にある場合は送信元ポート、to の次にある場合は送信先ポート番号にマッチします。またサービス名でも指定できます。

ping が使えないと悲しいので使えるようにします。

```
pass in on xl0 proto icmp all icmp-type 0
```

picmp-type ICMP メッセージのタイプコードを指定します。ping の場合エコー要求を通す必要があります。

```
pass out on lo0 all
pass in on lo0 all

pass out on xl0 all
pass in on xl0 proto tcp flags A/A
pass in on xl0 proto udp from any port = 53 to any
pass in on xl0 proto icmp all icmp-type 0
```

これでだいたいアクセスできるポートを制限できました。ftp を通したいなら ftp-data からの接続を受け入れられるようにする必要があります。しかし余計なポートを開けなくてはならないので、ftp はパッシブモードで使う方が良いと思います。

3 ちょっと本格的なフィルタ

これからちょとファイアウォールらしい設定をしてみます。
あやしげなパケットをはじきます。

```
block in from any to any with ipopts
block in proto tcp from any to any with short
```

1 行目 IP オプションが付いているパケットを叩き落す。
2 行目 短い TCP パケットを叩き落す。

断片化しているパケットは通常でもあるかもしれないので、とりあえずはじかないでおきます。
プライベートアドレスをはじきます。
これで IP アドレス偽装攻撃を防ぐことができます。

```
block in on xl0 from 10.0.0.0/8
block in on xl0 from 172.16.0.0/12
block in on xl0 from 192.168.0.0/16
```

あとマルチキャストもはじいてしまってよいでしょう。
内部から出ていくのも塞ぎます。

```
block out on xl0 from any to 127.0.0.1/8
block out on xl0 from any to 172.16.0.0/12
block out on xl0 from any to 192.168.0.0/16
```

NetBIOS なんか捨てます。

```
block out on xl0 proto tcp/udp from any port 136 >< 140 to any
block out on xl0 proto tcp/udp from any to any port 136 >< 140
block in on xl0 proto tcp/udp from any port 136 >< 140 to any
block in on xl0 proto tcp/udp from any to any port 136 >< 140
```

ここまでのをまとめます。

```
pass out on lo0 all
pass in on lo0 all

pass out quick on xl0 all
block out quick on xl0 from any to 127.0.0.1/8
block out quick on xl0 from any to 172.16.0.0/12
block out quick on xl0 from any to 192.168.0.0/16
block out quick on xl0 proto tcp/udp from any port 136 >< 140 to any
block out quick on xl0 proto tcp/udp from any to any port 136 >< 140


block in log quick on xl0 from 10.0.0.0/8
block in log quick on xl0 from 172.16.0.0/12
block in log quick on xl0 from 192.168.0.0/16

block in log quick on xl0 proto tcp/udp from any port 136 >< 140 to any
block in log quick on xl0 proto tcp/udp from any to any port 136 >< 140
```

```
pass in quick on xl0 proto icmp all icmp-type 0
block in log quick from any to any with ipopts
block in log quick proto tcp from any to any with short

pass in quick on xl0 proto tcp flags A/A
pass in quick on xl0 proto udp from any port = 53 to any
```

- log ログを記録するようになります。
- quick ルールは通常最後にマッチしたルールが有効になりますが、quick を付けるとその場で適用され、以降のルールは無視されます。ルールが複雑になってきた場合は、高速化のためにも使ったりします。

あとは必要なポートを開けていけばよいでしょう。

4 おまけ

IPv6 のルールをうまく一緒に書けないみたいなので、/etc/ipf6.conf に別に用意します。

```
#ipf -Fa -Z -f /etc/ipf.conf -6 -f /etc/ipf6.conf
```

これで IPv6 でのルールが設定できます。

今回は激しく時間がなかったのでほとんど未完成です。すいません。くわしいルールファイルの文法は ipf(5)、ipf のオプション等は ipf(8) を参照してください。

Another Nim

菊池 孝治 ⟨xmoe@mma.club.uec.ac.jp⟩

2002 年 11 月

1 序文

xmoe です。

今年もいつの間にか調布祭の季節となり、百万石の印刷も明日に迫ってしまいました。

人生こんなはずじゃなかったのに、などという、さよなら青い鳥的な精神状態であるわけですが[1]、編集担当者 (2) としては、このままでは遅筆な部員の方々にシメシがつかないという強迫観念もありますので、昨日購入した古本にヒントを得て (というかネタ本として使わせてもらって)、「2 倍取りゲーム」なるものについて書いてみたいと思います。

2 2倍取りゲームってなに?

某全国展開古本屋で購入した、ポケコン BASIC ゲーム本 [1] に掲載されていた石取りゲームです。

その本にはこんな説明がありました。

> このゲームはある数 (例えば n とします) から、あなたとコンピュータが交互に数を取っていって、最後の手で全部取った方が勝ちというゲームです。まず始めは、1 ～ n - 1 の間ならいくらでも取れます。つまり全部取らなければ、いくらでもよいのです。
>
> すると次に相手が取りますが、この回からは相手がとった数の 2 倍までしか取りません。相手が 2 をとったら 1 ～ 4 まで取れるのです。次はまた相手が取った数の 2 倍まで取れるのです。こうしてゲームを進めて行きます。

よくありそうな (というと失礼かもしれませんが) ゲームです。しかし、その記事には、「コンピュータは最良の手を打ってきます」という文章とともに、30 行足らず (コンピュータの思考ルーチンとして 10 行程) の BASIC プログラムが紹介されていました。(リスト 1)

そこで、僕の心に、「なんとかしてこのプログラムを理解してやるぜへいへい」という、将軍様の膝の位置はどういう仕組み?的な好奇心が湧いてきてしまったのです。

リスト 1: 掲載されていたプログラムのコンピュータの思考ルーチン部分 (一部改変)

```
A = 残りの石の数
B = 相手が前回取った数

100 C=A
110 N=C:GOSUB 200:IF C<>X THEN C=C-X:GOTO 110
120 IF C<=2*B THEN (C 個取るのが最良の手) ELSE (最良の手がないので、適当に取る)

200 X=1:Y=1
210 IF N<Y THEN RETURN
220 Z=Y:Y=Y+X:X=Z:GOTO 210
```

3 で、どうなりました? (by 円楽師匠)

正直な話、僕にはリストをみても、何が何だかさっぱりなので、自分でアルゴリズムを考えてみることにします。

石取りゲームと聞いたらテーブルを作って動的計画法[2]、という言葉があるかどうかは知りませんが、ちょっと考えてみると、方針としてそんな形でいけると思います。

win という整数配列を考えます。

win 配列は、

「ある数の石が残っていたときに、最小何個取れば必ず勝てるか」

[1] これは百萬石に限ったことではない

[2] 元の問題より小さいサイズの問題から順に解いていく考え方、らしいです。

を表す配列とします。

つまり、

win[n] = m

であれば、

石が n 個残っていたときに、自分が m 個取ることができれば勝ち

ということになります。

win 配列の中身は、win[1] を求めて、その結果を利用して win[2] を求めて、その結果を利用して win[3] を求めて ... という形で、簡単に求めていくことができます。

具体的に言うと、

1. 石が 1 つ残っていれば、1 つ取ることにより勝てるので 、win [1] = 1
2. 石が 2 つ残っていれば、1 つ取っても勝てない。なぜなら 1 つ取った時点で、残りの石は 1 つとなり、2 倍ルール上、相手は最大 2 つの石を取ることができる。ここで win[1] = 1 であるから、相手は 1 つの 石を取ることができれば勝てる。すなわち相手は勝ててしまい、結果自分は負けてしまう。また、自分が 2 つ取れば、その時点で勝ちとなる。よって win[2] = 2
3. 同様に、win[3] = 3
4. 石が 4 つ残っているとする。この時点で 1 つの石を取ると、3 つの石が残り、2 倍ルールにより、相手は 2 つまでの石を取ることができる。しかし win[3] = 3 であるから、相手は勝てない。すなわち自分が勝てる。よって win[4] = 1

ということで、上記のようなアルゴリズムにより、win 配列を求めるプログラムがリスト 2 です。
また、何をやっているのかはいまいち分からない、リスト 1 のような方法で win 配列を求めるプログラムがリスト 3 です。

二つのプログラムを実行してみると、一見同じ結果が得られているようです。(リスト 4)
一体なぜでしょう?

リスト 2: stone1.c

```
#include <stdio.h>
#include <stdlib.h>

#define STONE_MAX 20

int
main (void)
{
  int win[STONE_MAX + 1];
  int i, j;

  /* calc win[] */
  for (i=1; i<=STONE_MAX; ++i) {
    j = 1;
    while (j<i && win[i-j] <= j*2) {
      ++j;
    }
    win[i] = j;
  }

  /* display win[] */
  for (i=1; i<=STONE_MAX; ++i) {
    printf ("%d ", win[i]);
  }
  printf ("\n");

  exit (EXIT_SUCCESS);
}
```

```
#include <stdio.h>
#include <stdlib.h>
```

リスト 3: stone2.c

```
#define STONE_MAX 20

int
f (int a)
{
  int x, y, z;

  x = 1;
  y = 1;

  while (y <= a) {
    z = y;
    y = y + x;
    x = z;
  }

  return z;
}

int
main (void)
{
  int win[STONE_MAX + 1];
  int i, a, b;

  /* calc win[] */
  for (i=1; i<=STONE_MAX; ++i) {
    a = i;
    while (0<a) {
      b = f (a);
      a = a - b;
    };
    win[i] = b;
  }

  /* display win[] */
  for (i=1; i<=STONE_MAX; ++i) {
    printf ("%d ", win[i]);
  }
  printf ("\n");

  exit (EXIT_SUCCESS);
}
```

リスト 4: 実行結果

```
[xmoe@tp560x hyaku]$ ./stone1
1 2 3 1 5 1 2 8 1 2 3 1 13 1 2 3 1 5 1 2
[xmoe@tp560x hyaku]$ ./stone2
1 2 3 1 5 1 2 8 1 2 3 1 13 1 2 3 1 5 1 2
[xmoe@tp560x hyaku]$
```

4 fibonacci っておいしいですか?

と、ここまで考えてみても、リスト 1 が何をしているのかがさっぱりなわけで、ちょこっと google で検索したり、アルゴリズム辞典 [2] を紐解いたりしてみました。
結果、リストはどうも、"フィボナッチ分解"なることをしているようです。
フィボナッチ数列というものがありますね。皆さんご存知の通り、

$$n_1 = 1, n_2 = 1, n_i = n_{i-1} + n_{i-2} (3 \leq i)$$

という再帰的な形で定義されている数列です。最初の数個を並べてみると、

$$1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

といった感じになります。また、この数列の中に出現する数を、"フィボナッチ数" と言うらしいです。

で、"フィボナッチ分解" というのは、"整数を、一つ以上のできるだけ大きなフィボナッチ数の和として表す" ことらしいです。すなわち、

$$\begin{aligned}
&1 = 1 \\
&2 = 2 \\
&3 = 3 \\
&4 = 3 + 1 \\
&5 = 5 \\
&6 = 5 + 1 \\
&\ldots
\end{aligned}$$

ということらしいです。

で、アルゴリズム辞典には、`win[n]` を求めるためには、n をフィボナッチ分解したときに、一番小さな項を求めればよい、ということが書いてありました。なぜでしょう?

さて、今日も 9 時からコード書きのバイトがあったりするわけで、フィボナッチは来年までの課題、心あるかたは参考文献の中身を僕に噛み砕いて教えてくださいということで、眠りにつこうと思います。

ごめんなさい、ごめんなさい、ごめんなさい。

参考文献

[1] I/O 別冊：プログラム電卓ゲーム 2，工学社

[2] 奥村晴彦：C 言語による最新アルゴリズム辞典，技術評論社

MMAの1/4世紀 原稿メモ
MMA 創設から OS/9 前夜まで

中原 隆文 (ottan@mma.club.uec.ac.jp)
2002 年 11 月

1 いいわけ

部史及び活動報告誌を google りながら読み解きつつ作ったメモである。
1981 年までたどりついた時点で締め切りをむかえてしまったので、解説や脚注はいっさいなし。疑問、質問、追加情報は直接 ottan まで。
もしかしたら次号以降の百万石に完全版が載ることがあるかもしれない。

2 メモ

197504 創設メンバー入学

197512 同好会として創設
顧問は有山先生

1976 1 号機 (SM-390Z?)
(詳細不明 CPU 6800 ? / モトローラ 評価キット MEK6800 D II ベース ?)

197611 調布祭初参加

19761125 現存部史記述開始

197704 プロジェクト

- MMA SYSTEM 360
- マーガレット
- 16 進 keyboard
- 7 seg. LED display
- T.T.Y. (たぶんほんもののプリンタつきのテレタイプ)
- casette interface
- dot matrix display
- char. display
- MONITOR (マシン語モニタ?)
- ASEMB (マシン語アセンブラ?)
- HUNTER KILLER (オシロのゲーム?)

(19770428 部史記述に因る)

- カラー disp. 128x128
- モトローラ評価キット (1 号機 か M-360 のベース?)

(2001 合宿 取材に因る)

197704? 2 号機 (M-360 / マーガレット?) 製作
CPU MB8861 (富士通版 6800) / RAM 8K byte

197706? 部に昇格
(19770708 部史に meeting での報告の記述あり)

197711 調布祭 (予定)

- カラーゲーム (LKIT-8 + カラーディスプレイ)
- TK-ゲーム (TK-80 + TVD-01?)
- シンセ (1 ボード + シンセインターフェイス)
- お話、迷路 (マーガレット + TTY)

- BASIC (H68/TR + キーボード + TVD-02?)
- 講演 (安田 (安田寿明?)、石田 (石田晴久?) 両氏による講演)
- レーザー (シンセ研担当)

(実際のところは不明)
(翌年 4 月の記述に "Soft のはしる Hard を作れなかった" とあるのでソフト部分はあんまり出来なかった?)

197804 プロジェクト

- MAIN SYSTEM CPU XC6800 ? / RAM 32K byte (ES 版をどっかから手に入れた?)
- 32k RAM
- VRAM (キャラクタ 64x24)
- 256x256 グラフィック (モノクロ)
- Floppy or MT-2 (デジタルテープドライブ?)
- 別冊マーガレット CPU MB8861N / 2.25K byte
- Musical
- パーパート跳び亀
- START WARS GAME

(19780426/19780529 部史記述に拠る)

- FUJITSU LKIT-8
- H68/TR
- M/DOS

(2001 合宿 取材に拠る)

197811 調布祭

- MAIN SYSTEM
- GRAPHIC DISPLAY
- FLOPPY DISK
- 別冊マーガレット
- JOY STICK INTERFACE
- 自動演奏システム (TK-80(演奏) + H68/TR(音符表示) + シンセ)
- COSMO TERMINAL D (CPU MB8861 LKIT-8 ベース?)
- SNAKE (ハードは?)
- STAR WARS GAME (ハードは?)
- HALISP (MAIN 用 TINY LISP のこと?)

(MMA 臨時創刊号 に拠る)

1979 部史 及び 活動報告誌 現存せず

- デラマ?
- ALTAIR 880,680 ?

1980 プロジェクト

- M/EDIT (ラインエディタ?)
- MAC,LINK,MERGE (6800 アセンブラツール)
- DEBUG MONITOR (6800 デバッガ?)
- PLAN 68/09 (PASCALっぽい言語)
- HALISP (LISP)
- ALTAIR BASIC (BASIC)
- TL1 (TL/1 TL は TanyLanguge の略らしい PL/1 もどきか ?)
- M.M.M. (MMA マイクロマウス 迷路を走るやつ)
- P.S.S. (Programable Sound Source シンセ関係?)

- デラックスマーガレット (CPU MB8861N, RAM 68 k Byte)
  (1979 には製作されていた模様だが、故障している期間が長く休眠状態だった模様)

(部史 (198011-198002) に因る)
また、部史 8012271442, 8101021912 あたりに、6809 用の PLAN 09 や逆アセンブラのはなしが出てるので、1980 末にはデラマに 6809 が追加されていたか、別に 6809 マシンも稼働していた模様用

19800620 部史再開

198011 調布祭

- MMA Soft Ware 開発システム (EDIT, MAC, LINK)
- グラフィックディスプレイ
- MMA DBMS
- M/DOS 68
- MMA "Delaxe Margaret" (CPU MB8861N, RAM 68 k Byte)
- RTTY 受信とその解読
- 3 次元グラフィックスとフライトシミュレーション
- MMA MAIN SYSTEM 解説

(MMA Vol.3 1980 に拠る)

1981 プロジェクト

- MMM
- 68k system
- 68k シュミレーター
- 68k アセンブラ
- 68k モニター
- PSS
- PLAN 68000 cross compiler
- S-100 system (バスの名前じゃ?)

(部史 (1981) に因る)

- CP/M 86 (川島)

(2001 合宿 取材に拠る)

198104 新歓 (予定)

- マンネンカレンダー
- バイオリズム
- ペントミの
- 地底最大の作戦
- プリンター関連
- ブレークアウト
- カタログシステム
- デラマ

(シンブシ 198103201420 に拠る)
部史の記述を見るとこの年には既に新宿からの行軍を行っている模様

198107 プレハブの部室へ移転

198111 調布祭

- "FIFIDUS MIL-MIL" THE SYSTEM OF 68000
  (MPU HD68000L8 / RAM 128 KB ソフトはまだ簡易モニタのみ)
- "GIGANT" MMA MICROMOUSE II
  (第 2 世代マイクロマウス)
- M/DOS68 THE MMA ORIGINAL FDOS

- PROBRAMMABLE SOUND SOURCE
  (MB8870(MC6802 相当) + AY-3-8910 PSG 音源 MAIN SYSTEM 上で MML をコンパイルして流し込む)
- "WHISKY" ON TRAINING SYSTEM
  (印刷状態が悪く詳細不明)
- "MESSIAH" ON EVALUATION BOARD OF i8088 (MMA ではめずらしい 80 系)
- AN INTRODUCTION OF COMPUTER SCIENCE
  (書籍案内)

(MMA Vol.4 '81 に拠る)

編集後記

xmoe ⟨xmoe@mma.club.uec.ac.jp⟩

1　学友会の人が

怒ってしまうと恐いので、そろそろ印刷しにいってきます。おみぐるしい点はごめんなさい。

2002 autumn

百萬石 chofu-fest edition